Batchsize = 256

Lr = 0.007

Epoch = 300

final training accuracy: 0.778983526605404 final training loss: 0.03980357964572154

final validation accuracy: 0.7064282980689122 final validation loss: 0.0852205018912043

final test accuracy: 0.7086113183805258 final test loss: 0.08060089924505778

class Autoencoder\_3\_0(nn.Module):

def \_\_init\_\_(self, dropout\_rate=0.2):

super(Autoencoder\_3\_0, self).\_\_init\_\_()

# Encoder

self.enc1 = nn.Sequential(

nn.Conv2d(3, 16, 16, stride=2, padding=1),

nn.ReLU(True),

nn.MaxPool2d(2, stride=2, padding=0),

)

self.enc2 = nn.Sequential(

nn.Conv2d(16, 32, 5, stride=2, padding=1),

nn.ReLU(True),

nn.Dropout(dropout\_rate),

nn.Conv2d(32, 64, 3, stride=2, padding=1),

nn.ReLU(True),

)

self.enc3 = nn.Sequential(

nn.Conv2d(64, 128, 3, stride=2, padding=1),

nn.ReLU(True),

nn.MaxPool2d(2, stride=2, padding=0),

)

self.dec1 = nn.Sequential(

nn.Upsample(scale\_factor=2, mode='nearest'),

nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1, output\_padding=0),

nn.LeakyReLU(True),

)

self.dec2 = nn.Sequential(

nn.ConvTranspose2d(64, 32, kernel\_size=4, stride=2, padding=1),

nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output\_padding=1),

nn.LeakyReLU(True),

nn.Dropout(dropout\_rate),

)

self.dec3 = nn.Sequential(

nn.ConvTranspose2d(16, 16, kernel\_size=4, stride=2, padding=1),

nn.ConvTranspose2d(16, 3, 16, stride=2, padding=1, output\_padding=0),

nn.LeakyReLU(True),

nn.Sigmoid(),

)

def forward(self, x):

# Encoder

enc1 = self.enc1(x)

enc2 = self.enc2(enc1)

enc3 = self.enc3(enc2)

dec1 = self.dec1(enc3)

# Resize and add skip connection from enc2

dec1 = F.interpolate(dec1, size=enc2.size()[2:]) + enc2

# Assuming dec2 corresponds to the decoder layer after dec1

dec2 = self.dec2(dec1)

# Resize and add skip connection from enc1

dec2 = F.interpolate(dec2, size=enc1.size()[2:]) + enc1

# Assuming dec3 corresponds to the final decoder layer

x = self.dec3(dec2)

return x